U.S. ARMY
CAA-2017013

Center for Army Analysis

MARATHON VERIFICATION (MARV)

AUGUST 2017

CENTER FOR ARMY ANALYSIS
6001 GOETHALS ROAD
FORT BELVOIR, VA 22060-5230

REPORT DOCUMENTATION PAGE		*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* August 2017	**2. REPORT TYPE** Final	**3. DATES COVERED *(From –To)*** Oct 2016 – Jul 2017
4. TITLE AND SUBTITLE MARATHON Verification (MARV)		**5a. CONTRACT NUMBER**
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Rick Hanson Tom Spoon		**5d. PROJECT NUMBER**
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Center for Army Analysis** **6001 Goethals Road** **Fort Belvoir, VA 22060-5230**		**8. PERFORMING ORGANIZATION REPORT NUMBER** CAA-2017013
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Center for Army Analysis 6001 Goethals Road Fort Belvoir, VA 22060-5230		**10. SPONSOR/MONITOR'S ACRONYM(S)**
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
DISTRIBUTION A. Approved for public release: distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The Modeling Army Rotation At Home Or Not (MARATHON) simulation helps analyze the dynamic consequences of force generation policies governing supplies of force structure readied for employment against demands. This study is a comparative verification of the functionality of MARATHON 4 (our newest implementation of MARATHON) to MARATHON 3 (our legacy implementation). They were developed in two different runtimes and have different code bases. We discuss an approach for the verification and "lessons learned" from the semantic and technical issues we discovered as we implemented the approach.

15. SUBJECT TERMS
MARATHON, verification, comparative verification, V&V, testing, modeling, simulation, force generation, ARFORGEN, rotational analysis

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Mr. Russell Pritchard
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UU	**36**	**19b. TELEPHONE NUMBER *(include area code)*** (703) 806-5421

(THIS PAGE INTENTIONALLY LEFT BLANK)

MARATHON VERIFICATION (MARV)

SUMMARY

THE PROJECT PURPOSE
This study verifies the correct functionality of the Modeling Army Rotation At Home Or Not (MARATHON) simulation version 4 (“M4”) relative to MARATHON version 3 (“M3”).

THE PROJECT SPONSOR
Dr. Steven Stoddard, Center for Army Analysis (CAA)

THE PROJECT OBJECTIVES were to:

(1) Demonstrate that the M3 and M4 simulations (computer implementations of the MARATHON model) are indeed functionally equivalent.

(2) Document the verification process with results and report any issues.

THE SCOPE OF THE PROJECT
We confined the verification of the functionality of M3 to comparative verification.
M4 has more functionality than M3 (M4 models Sustainable Readiness in part, for instance); however, we only compared the subset of M4’s functionality that corresponds to the functionality of M3. (Namely, these were the capacity analysis and requirements analysis functionalities.) Also, the test input datasets for the verification (by comparison) were from the Total Army Analysis (TAA) cases from TAA 20-24.

THE MAIN ASSUMPTIONS
The machine execution of the algorithms of M4 is identical to those of M3 — at the time of the study — and the tools that we used for the output comparison were working correctly.

THE PRINCIPAL FINDINGS are:
We achieved the study objective of demonstrating functional equivalence, but not without adjusting both model implementations in order to shore up small bugs and inconsistencies.

THE PRINCIPAL RECOMMENDATIONS are:

(1) Declare that M4 is the model of record for all force structure analyses as of July 2017.

(2) Continually verify the MARATHON model (i.e., M4 and beyond) tied to regression/unit tests and other techniques such as generative testing.

THE PROJECT EFFORT was conducted by Rick Hanson and Tom Spoon, both members of CAA’s Force Strategy Division.

COMMENTS AND QUESTIONS may be sent to the Director, Center for Army Analysis, ATTN: CSCA-FS, 6001 Goethals Road, Suite 102, Fort Belvoir, VA 22060-5230.

(THIS PAGE INTENTIONALLY LEFT BLANK)

CONTENTS **Page**

FIGURES

(THIS PAGE INTENTIONALLY LEFT BLANK)

1 INTRODUCTION

Force Generation – the preparation and utilization of Army force structure for deployment to meet contingency operations - has long been analyzed at the Center for Army Analysis (CAA). Simulations developed by analysts in the Force Strategy (FS) Division of CAA are the central tools for force structure analysis, with the Modeling Army Rotation At Home Or Not (MARATHON) simulation serving the backbone of the FS force structure analysis capability.

MARATHON-based analyses answer the following questions:

- "How large a force do I need, and with what capability mix?"
- "How much can I do with the force I have, or expect to have?"

The FS Division has conducted MARATHON-based analyses for organizations such as the National Commission on the Future of the Army (NCFA), Quadrennial Defense Review Office, Headquarters, Department of the Army G-3/5/7, Chief of Staff of the Army, and Joint Staff.

1.1 The Origin of MARATHON

In 2004, CAA developed the MARATHON simulation to analyze the dynamic consequences of force generation policies governing supplies of force structure readied for employment against demands, with temporally variable supplies, demands, and policies.

MARATHON started as a mechanism to analyze the dynamics of Army Force Generation (ARFORGEN), and to explore the vast realm of unintended consequences for which the static analysis could not account. Analysts at CAA implemented an ARFORGEN-based simulation in the proprietary ProModel simulation platform as a proof-of-concept. As the centers of gravity of the Army, brigade combat teams (BCTs) were the initial focus for rotational analysis. The early set of capabilities were intentionally limited and scoped to facilitate rapid development and prototyping. BCTs flowed from one location to the next in their ARFORGEN cycles, checking for unfilled deployments when feasible.

1.2 MARATHON Versions 1, 2 and "3"

The success of the early "proof of concept" led to vigorous development and usage of MARATHON, resulting in the first production version: MARATHON 1. ProModel continued to serve as the primary logic, parameter, and visualization platform for MARATHON 1, with some external data residing in Microsoft Excel worksheets. Due to its use in many operations research curricula, analysts looking to simulate ARFORGEN were familiar with ProModel and had buy-in from CAA. The simulation platform proved suitable for rapid prototyping of simple systems (e.g., BCTs) due to prefabricated design tools, a modicum of data structures, a scripting language, and decent integration with Excel.

In 2005, MARATHON 1 went through verification, validation and accreditation (VV&A) by way of a validation case study provided by Dr. W. F. Crain, DAMO-CI.

The next generation design, MARATHON 2, simultaneously sought to solve the performance and content management problems accumulated through earlier generations of MARATHON. The new design proposed changes to the fundamental algorithms for selecting units for deployment, particularly an order-of-magnitude reduction in the number of times units underwent fill-selection queries. The design also called for a near-total separation of data from logic, to facilitate a data-driven simulation approach and to facilitate large numbers of varying runs. Accomplishing both of these tasks required significant redesign of the underlying architecture and internal data representation present in MARATHON 1.

MARATHON 2 used a comparative verification approach, but without the rigor of maintaining identical outputs. Since both the input and output data models grew, in addition to performance improvements and architectural changes, MARATHON 2 was only relatively comparable with MARATHON 1 in terms of output. Rather, the MARATHON 2 verification cases were designed to ensure correct implementation of the new algorithms. MARATHON 2 entered production in June 2009 after successful MARATHON 2 internal verification cases and a successful coarse comparison of MARATHON 1 and MARATHON 2 output.

Due to a limitation of MARATHON 2, specifically the inability to extend or otherwise address limitations in ProModel, MAJ Alex MacCalman — with help from Mr. Tom Spoon — initiated a regressive, scoped-down port of MARATHON functionality to Excel Visual Basic for Applications (VBA). This version, dubbed MARATHON "3," simultaneously advanced and regressed the state of MARATHON's features. Away from ProModel's closed-source limitations, MARATHON "3" opened up the possibility for custom-built libraries and data structures, as well as leveraging existing facilities such as the Solver optimization model and numerous Excel libraries.

As before, MARATHON "3" reverted to a simplified BCT-centric simulation aimed at a narrow use case (supporting the ARFORGEN Availability Exploration Study [AAES]), disregarding a feature-complete port of MARATHON 2 in favor of an expedient piece of research software to support a priority study. Most notably, MARATHON "3" underwent no formal verification — comparative or otherwise. Consequently, MARATHON "3" left MARATHON development with a loss of version control and multiple model management problems: MARATHON "3" failed to implement the functionality from MARATHON 2 required for Total Army Analysis (TAA) and regressed to an inefficient, poorly scaling algorithmic basis for simulation, yet added some novel features required for future efforts.

1.3 MARATHON Version 3 – M3

MARATHON 3 ("M3") was Tom Spoon's 2010-2012 rewrite of the initial limited 2010 VBA port of MARATHON 2 by (then) MAJ Alex MacCalman. M3 was developed and used in studies from the time of the rewrite, up through the TAA effort conducted in 2016 (TAA 20-24).

M3 unified the disparate features of MARATHON 2 and MARATHON "3", retaining the benefits of MARATHON 2 (an efficient event-based simulation and the ability to compute force structure requirements for TAA) and the benefits of MARATHON "3" (an extensible VBA platform vs. ProModel and portability via Excel). Due to input, output, and implementation differences between M3 and its precursors, comparative verification provided limited utility. The initial 2010-2012 Army Rotational Modeling Blueprint for Advancement and Redesign study augmented limited comparative verification with dynamic verification methods, unit-tests, and subject matter expert testing to verify the M3 implementation. Force structure analysts within the Force Strategy division and the TAA community served as beta testers and independent verifiers up to the production release of M3 during the spring of 2012.

Changing analytic demands largely drove M3's capability development and increased the scale of existing simulation infrastructure. High-profile TAA, NCFA, and G-3/5/7 Army War Plans Division work demanded larger experimental designs, boutique visualizations, and dramatically increased the processing and analytic workload. The Force Mix Composition Analysis study pushed the rotational analysis state-of-the art by adding stochastic demand futures, multiple sizable simulation runs, and additional business rules for resource-constrained force structure allocation.

Sustainable Readiness (SR) and visualization also drove emerging capability requirements for M3. SR continues to overturn legacy readiness assumptions and business rules, requiring flexible readiness policies, variable business rules, unique entity behaviors, and extensive supply-demand relations. Senior leaders' demand for animation and interactive visualization has created the need for reifying simulation history into compelling "real-time" visuals.

M3 demonstrated technical limitations during its production usage in the aforementioned studies and projects, clearly evincing the need for a new technology. These limitations included the lack of flexibility in data structures and control flow, Excel VBA memory leaks for TAA requirements analysis runs, inability / burdens on performing large-scale simulation runs, and unnecessary dependence on Excel. Hence, they undermined the ability to address sponsor needs across multiple areas — stochastic demand futures in force composition analysis, large scale design of experiments, Sustainable Readiness, etc.

1.4 MARATHON Version 4 – M4

The profound technical limitations of VBA prompted the implementation of MARATHON 4 ("M4") to meet emerging and future requirements. Tom Spoon initiated the development effort in 2012, concurrent with late M3 development and other model development.

As a radical departure from previous ProModel and VBA implementations, we wrote M4 in Clojure, so it runs on the Java Virtual Machine (JVM), and it emphasizes scalable performance for large-scale analyses, dynamic visualization, and flexible development. It addresses the technical limitations of M3, while building a platform for in-demand advanced analyses.

The reader should consult the Army Rotational Modeling Blueprint for Advancement and Redesign 2017 documentation for a more detailed treatment of the development history of MARATHON.

1.5 Problem Statement, Scope, and Assumptions

The issue at hand, which motivated this study, was that we had not verified M4 to be functionally equivalent to M3. This was necessary so that M4 could serve as the primary simulation platform for force generation analysis going forward, and importantly, would mitigate the costs of resourcing model development maintenance and support effort for two models at the same time. Before initiating this study, the FS Division had to resource for both model implementations.

In this study, we verified the correct functionality of M4 relative to M3, that is, by direct comparison. We confined the verification of the functionality of M3 to comparative verification. M4 has more functionality than M3 (for instance, M4 models Sustainable Readiness in part, whereas M3 does not at all); however, we only compared the subset of M4's functionality that corresponds to the functionality of M3. (Namely, these were the capacity analysis and requirements analysis functionalities.) Also, the test input datasets for the verification (by comparison) were from the production cases from the TAA 20-24 study.

We identified two basic assumptions upon which this effort depended: (1) that the machine execution of the algorithms of M4 would be identical to those of M3, and (2) that the tools that we used for the output comparison would be working correctly.

The first assumption means that the machine, either the Excel VBA platform in the case of M3 or the JVM in the case of M4, will faithfully execute the algorithms we write in the source code of the respective computer languages, that is, the machines would work as stated in the manuals and references provided by the vendor. When this does not hold, it means that there is one or more bugs in the implementation of the machine itself. Since we don't control the configuration of these machines, "all bets are off," and this would entail that we could not trust the model implementations or their outputs.

Related to the first assumption, we note that the hardware and operating system platforms upon which the virtual machines ran remained invariant between the M3 and M4 runtimes — only the aforementioned virtual machines were different.

If the second assumption did not hold, the study results in general could not be trusted. This assumption was necessary because we could not afford to embark on the *reductio ad absurdum* trail of "verifying the verifier."

For both assumptions, we may or may not have been able to detect the adverse condition.

Finally, within the scope of the study, we could not identify any study limitations that we could attributed to a lack of time, people, data, or methodology.

2 APPROACH

Before we discuss the approach per se, we discuss the current verification processes that we have in place for both implementations and the special conditions that prevailed that made advantageous the approach that we chose.

2.1 Extant Verification Processes

Regardless of this study's verification, at the time of this study, both M3 and M4 had multiple dynamic verification processes already in place.

They both employed **assertion checking**, which means that there are punctuated points in the source code (which correspond to moments during the runtime of the implementation) where certain conditions are checked. These particular conditions are sometimes called "invariants" and are in the form of "assertions", which are conditional expressions in the programming language that evaluate to `true` or `false` (i.e., Boolean) values during the runtime. When the "assertions" fail, i.e., when they evaluate to `false`, the running model (program) is halted and the state of the program that directly affects the failed assertion is logged (i.e., written to output) so that it can aid the user or developer in troubleshooting the issue at hand.

Both simulations employ mechanisms for **logging / observation**. At a minimum, M3 and M4 generate a common "event history," that is a textual log of all the events processed during the simulation. Additionally, domain-specific logs for deployments, demand fills, and other canonical outputs provide a common set of observable history for different domains of the simulation. Finally, both simulations provide facilities for interactively debugging or otherwise inspecting the simulation context as it progresses.

Either using the Proc post-processor (M3/M4) or by visualizing entities during a "live" run (M4), both simulations provide dynamic **animation / visualization / run-time plots**.

M4 has additional automated testing facilities thanks to the `clojure.test` library, which provides automated regression testing. M4's developer has already accumulated 48 unit tests that exercise the functionality of the entire system. The testing facility is command-line executable to facilitate continuous integration on a test server for "continual verification." Finally, tests are trivially extended to account for new cases, and – most importantly – repeatable.

2.2 Preconditions of the Approach

For this comparative verification, the models of interest meet all of the following conditions:

- Output file formats are identical.
- Intermediate computations are identical.
- Numbers and arithmetic operations have the same bounds and precision.

Because the developer of M4 ported M3 from the beginning, down to performing the same arithmetic computations and rendering the outputs in the same format, we only needed to compare the two sets of outputs textually.

This was our initial foundation and, as we will see later in the discussion of the results, some of these conditions were not quite true, but in a sense "almost true" and correctable to the exact precision when adjustments (fixes) to the model implementations were allowed.

2.3 Description of the Approach

We identified a collection of input datasets (an *input dataset* is a set of inputs for one case) with which we, on a case-by-case (dataset by dataset) basis, ran through both the M3 and M4 implementations and checked for the outputs to be textually equivalent. For two sets of output to be *textually equivalent*, the output files must be identical character-for-character.

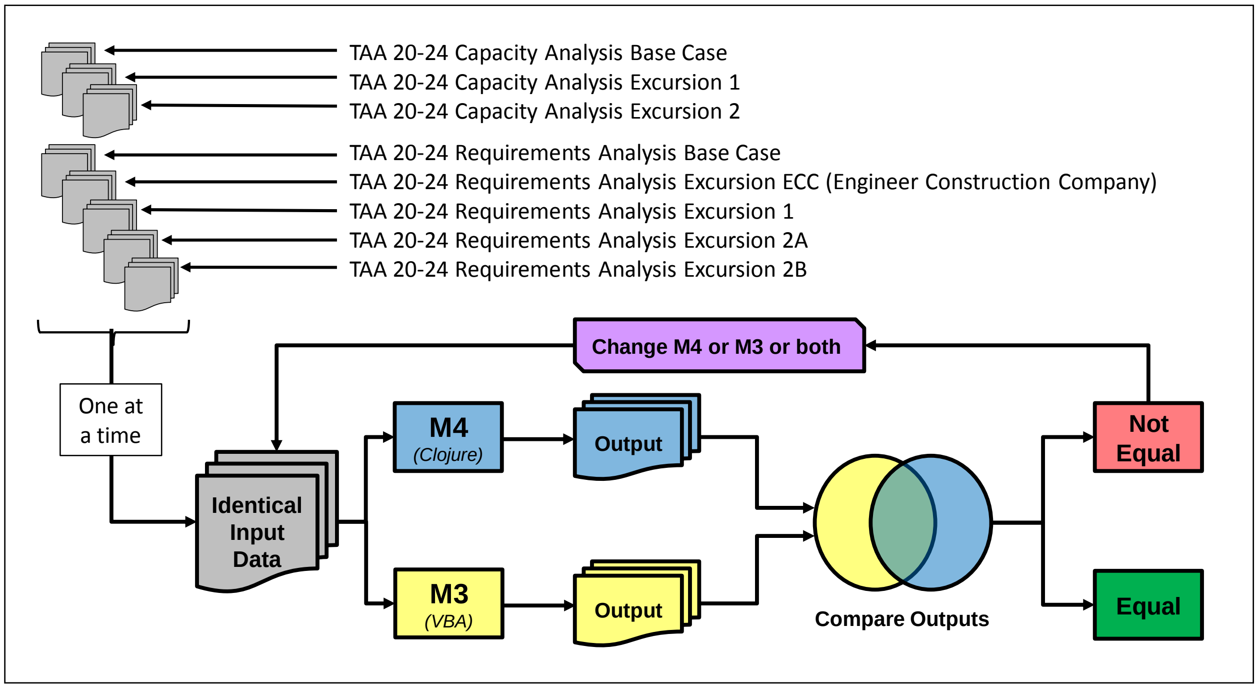

TAA 20-24 Capacity Analysis Base Case
TAA 20-24 Capacity Analysis Excursion 1
TAA 20-24 Capacity Analysis Excursion 2
TAA 20-24 Requirements Analysis Base Case
TAA 20-24 Requirements Analysis Excursion ECC (Engineer Construction Company)
TAA 20-24 Requirements Analysis Excursion 1
TAA 20-24 Requirements Analysis Excursion 2A
TAA 20-24 Requirements Analysis Excursion 2B
One at a time
Identical Input Data
Change M4 or M3 or both
M4 (Clojure)
M3 (VBA)
Output
Output
Compare Outputs
Not Equal
Equal

Figure 1. Diagram of the Approach.

Figure 1 shows the idea diagrammatically. For each input dataset — they were from among each of the TAA 20-24 cases for both the capacity and requirements analyses — we ran both M3 and M4 against it, producing raw output datasets that we then compared textually. If they ***matched – were textually equivalent -*** then we achieved the success condition, i.e., we succeeded in demonstrating the functional equivalence of the models with that input dataset. If not, we investigated what was causing the discrepancies, resolved the issue (e.g., fix the implementation(s) if a bug), then re-ran the case(s) that failed to match and repeated the raw output comparison.

We also developed scripts that compare the output datasets. We call these *diff* scripts. The idea was that we should be making the process of comparison as reproducible and automatable, and thus as transparent to audit, as possible.

We also anticipated accounting for floating-point (FP) "noise" introduced in all the intermediate calculations of FP operations because we did not know beforehand — and without performing a detailed scan of all the source code before we started — if all of the intermediate calculations were strictly carried out in exactly the same way between M3 and M4 (regardless of the algebraic equivalence of any aggregate calculations). In that case, we were prepared to relax the definition of *match* (bolded above) from *textually equivalent* to *essentially textually equivalent,* which we take to mean character-by-character equivalence except in the case of numbers, which were allowed to differ only in absolute value by ε, a predetermined error. We wrote the *diff* scripts to account for this. (We remind the reader that the scripts themselves have been assumed to work correctly.)

However, as we shall see in the discussion later, this would not turn out to be necessary as (1) there were very few FP operations in both M3 and M4, and (2) while there were discrepancies involved in FP operations that caused non-matching output, they did not manifest as numeric output errors differing by an absolute value greater than or equal to ε, as we had anticipated.

2.4 Analysis of the Test Design Space

We did not have time to widen the input dataset domain to cover more test cases nor perform an experimental design of the test inputs. Such an undertaking would require the study director to understand the model (and both implementations) past the neophyte level. Beyond that, additional time would be required to (1) do the further research to create a robust design, (2) generate the input datasets according to the design, (3) carry out the runs of the numerous excursions in both implementations, and (4) troubleshoot and fix implementation issues in the comparison feedback loop mentioned above.

To highlight the magnitude of this issue, we computed an upper bound of the test space of all inputs governed by the bounds of the TAA datasets. To do this, we started by enumerating the supply combinations per standard requirements code (SRC) and then the demand combinations per SRC. We then combined both combinations in turn and factored in the maximum number of SRCs.

The number supply combinations, per SRC, were bounded above by 552, since there were three force structure components (Active (AC), National Guard (NG), and Army Reserve (RC)), each of which could achieve a maximum supply quantity of 184.

The number of demand combinations, per SRC, was much larger - in order of magnitude - than the number of supply combinations for the same SRC. They were comprised of many more factors with more levels: two demand categories (rotational and non-Boots On Ground [NonBOG]), two ways to source the demand (either selecting uniformly or by selecting NG units first), the number of units in the SRC demanded (of which the maximum was 15), the day the demand started (which varied from Day 1 to Day 4,563), the duration of the demand in days (of which the maximum was 4,745), and finally, the number of times a demand record occurred for the SRC in question (whose maximum was 117). This yielded an upper bound of $151{,}993{,}073{,}700 \approx 1.51 \times 10^{11}$ demand combinations per SRC.

We found the total number of supply-demand combinations, per SRC, to be $83{,}900{,}176{,}682{,}400 \approx 8.39 \times 10^{13}$, and the number of TAA-relative combinations (i.e., with all SRCs considered — there are 317 of them) were $26{,}596{,}356{,}008{,}320{,}800 \approx 2.66 \times 10^{16}$.

Holding policy-related factors constant, the TAA-sized capacity analyses posed an enormous design space according to a naïve enumeration of the full-factorial experimental design. By analogy, if our task were to use a tablespoon to drain a bucket containing 422 times the water volume of Atlantic Ocean, we would have just drawn the first tablespoon by running one TAA case.

While there may have been significant pruning opportunities, these were not necessarily free or straightforward to compute. For the design space we encountered, adaptive sampling or sparse experimental designs appeared to be crucial to defining tractable, large-scale verification efforts.

3 APPLICATION

3.1 Detecting and Resolving Differences

The simple representation of "comparing outputs" and "change M4 or M3 or both" in the feedback loop of the study approach (depicted in Figure 1) abstracts a sophisticated process of model refinement. Model refinement (depicted in Figure 2) involves detecting differences, identifying divergent histories, reasoning about the divergence, and applying patches via the "Git" Distributed Version Control System (DVCS). The refinement process leverages automation (via the **marathon.vnv** Clojure scripts) and software-assisted interactive forensic analysis (via Excel workbooks and the Clojure Read-Evaluate-Print Loop [REPL]) to provide a reproducible process for finding differences in output, isolating the forensic context for possible causes of said differences, and refining one or both models to eliminate the differences.

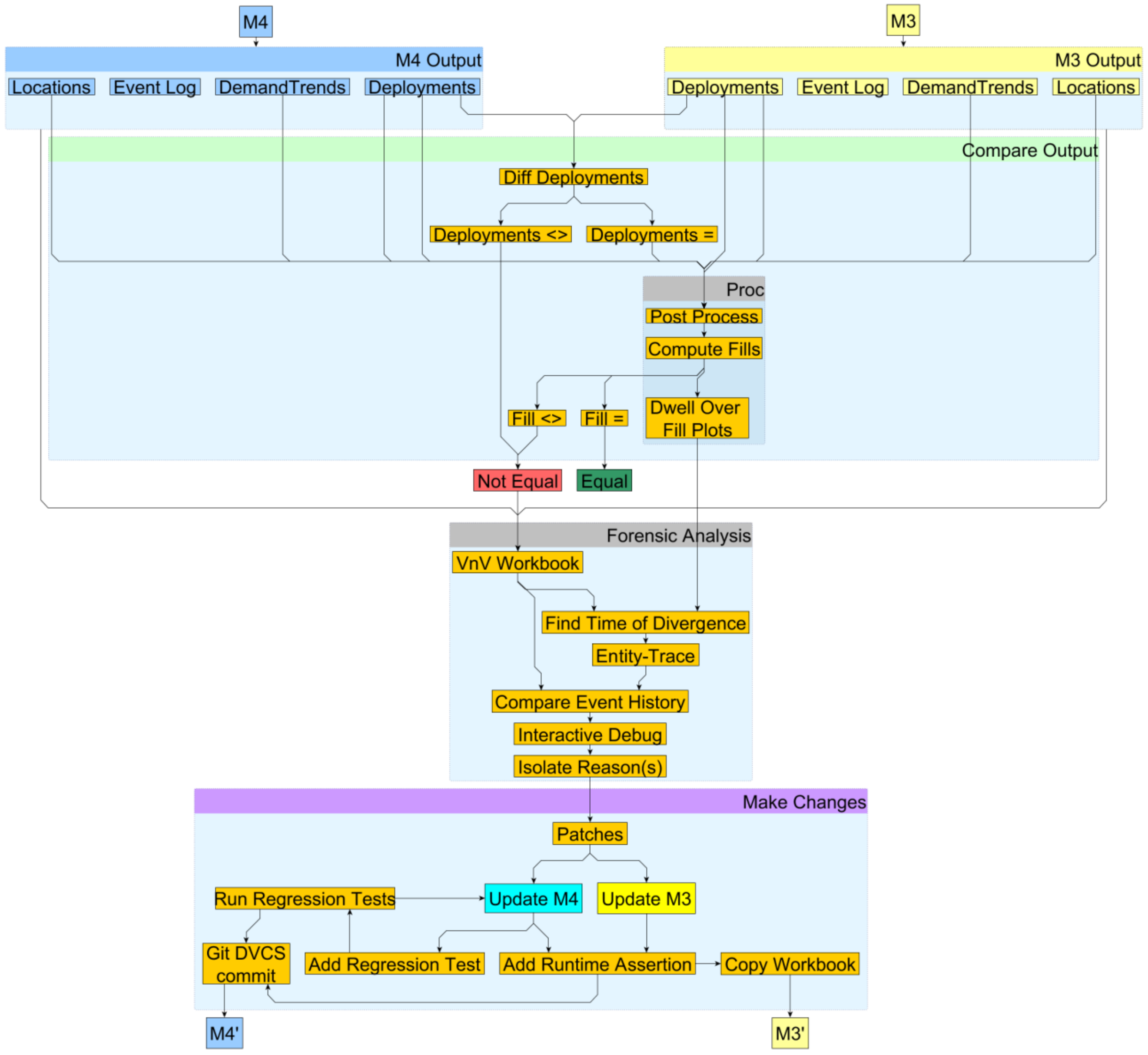
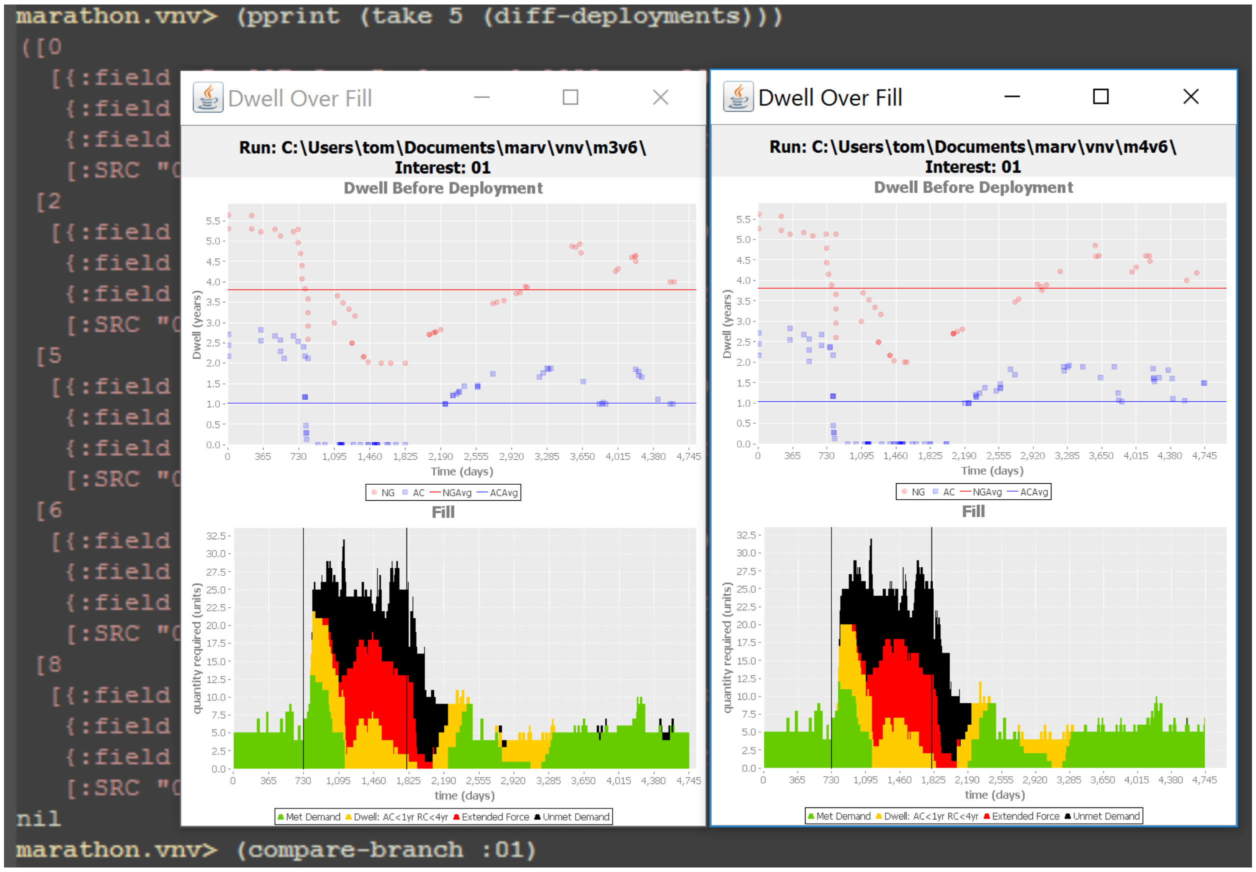


Figure 2. The Process of Model Refinement.

When we used the Clojure *diff* tool and Proc to check for differences, if the Clojure *diff* tool detected a difference, it was reported textually; then the verifier would process the outputs through Proc to visualize differences across the entire timeline as in Figure 3.



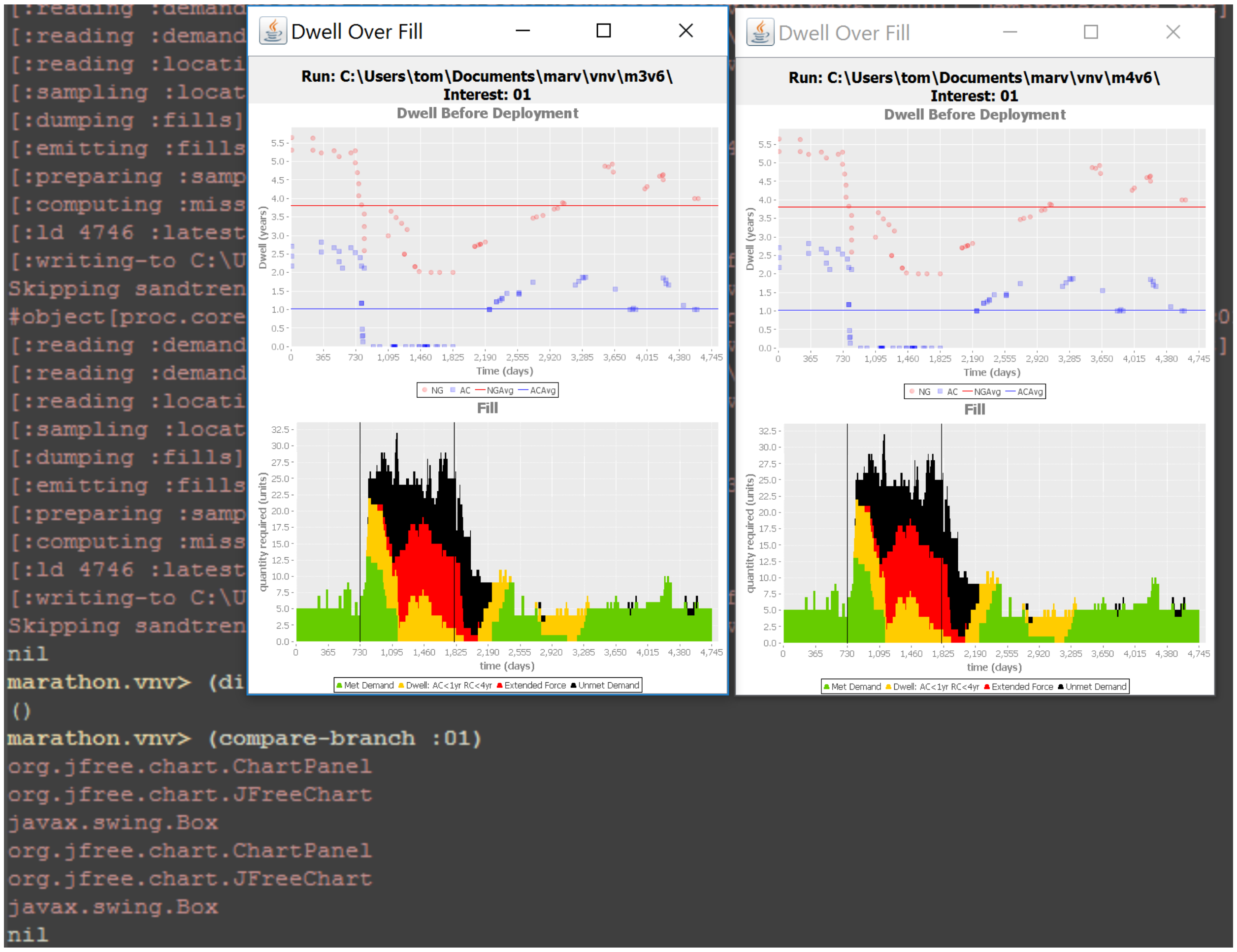
```
marathon.vnv> (pprint (take 5 (diff-deployments)))
([0
  [{:field
    {:field
    {:field
    [:SRC "
 [2
  [{:field
    {:field
    {:field
    [:SRC "
 [5
  [{:field
    {:field
    {:field
    [:SRC "
 [6
  [{:field
    {:field
    {:field
    [:SRC "
 [8
  [{:field
    {:field
    {:field
    [:SRC "
nil
marathon.vnv> (compare-branch :01)
```

Dwell Over Fill
Run: C:\Users\tom\Documents\marv\vnv\m3v6\
Interest: 01
Dwell Before Deployment
Dwell (years)
Time (days)
NG AC NGAvg ACAvg
Fill
quantity required (units)
time (days)
Met Demand Dwell: AC<1yr RC<4yr Extended Force Unmet Demand
Dwell Over Fill
Run: C:\Users\tom\Documents\marv\vnv\m4v6\
Interest: 01
Dwell Before Deployment
Fill

Figure 3. Textual and Visual *diff*ing Using marathon.vnv and Proc.

When a particular difference was resolved, the verifier would patch the model(s) and re-run that particular input case. The textual *diff* would typically reduce (that is, would show fewer differences) and the Dwell over fill charts as in Figure 3 would exhibit fewer differences.

Finally, after many rounds of detecting differences and fixing/adjusting the model(s), the textual *diff* would report no differences (in Clojure, the value “returned” would be an empty sequence) and the Dwell over fill charts between M3 and M4 outputs would be identical. See Figure 4 to see an example of what this condition looked like.


Dwell Over Fill
Run: C:\Users\tom\Documents\marv\vnv\m3v6\
Interest: 01
Dwell Before Deployment
Dwell (years)
Time (days)
NG AC NGAvg ACAvg
Fill
quantity required (units)
time (days)
Met Demand Dwell: AC<1yr RC<4yr Extended Force Unmet Demand
Dwell Over Fill
Run: C:\Users\tom\Documents\marv\vnv\m4v6\
Interest: 01
Dwell Before Deployment
Dwell (years)
Time (days)
NG AC NGAvg ACAvg
Fill
quantity required (units)
time (days)
Met Demand Dwell: AC<1yr RC<4yr Extended Force Unmet Demand

```
[:reading :demand
[:reading :demand
[:reading :locati
[:sampling :locat
[:dumping :fills]
[:emitting :fills
[:preparing :samp
[:computing :miss
[:ld 4746 :latest
[:writing-to C:\U
Skipping sandtren
#object[proc.core
[:reading :demand
[:reading :demand
[:reading :locati
[:sampling :locat
[:dumping :fills]
[:emitting :fills
[:preparing :samp
[:computing :miss
[:ld 4746 :latest
[:writing-to C:\U
Skipping sandtren
nil
marathon.vnv> (di
()
marathon.vnv> (compare-branch :01)
org.jfree.chart.ChartPanel
org.jfree.chart.JFreeChart
javax.swing.Box
org.jfree.chart.ChartPanel
org.jfree.chart.JFreeChart
javax.swing.Box
nil
```

Figure 4. No Textual or Visual Differences.

3.2 Numerical Issues

For the most part, we were able to avoid the typical FP issues between especially different implementations of the simulation models, as M3's and M4's arithmetic operations are almost all integer operations. However, there are a handful FP operations and one of them caused a positive difference in the model output comparison.

VBA has systemic, yet subtle inconsistencies, particularly with truncating FP numbers to get integers. Per the following examples:

- ? 6 / 10 $\xrightarrow{\text{yields}}$ 0.6 Correct
- ? 0.6*10 $\xrightarrow{\text{yields}}$ 6 Correct
- ? Fix(0.6) $\xrightarrow{\text{yields}}$ 0 Correct
- ? Fix(1.6) $\xrightarrow{\text{yields}}$ 1 Correct
- ? Fix(0.6*10) $\xrightarrow{\text{yields}}$ 5 (Incorrect, expected 6)

- ? Fix(0.4*10) $\xrightarrow{\text{yields}}$ 4 (but this one is correct!)
- ? Fix(0.3*10) $\xrightarrow{\text{yields}}$ 2 (ah, wrong again … expected 3!)

The solution rested in defining custom cross-platform FP truncation functions in both M3 and M4. These functions created a consistent means for truncating decimals to six decimal places. Through testing, we found additional edge-cases for VBA that caused errors in FP truncation or counter-intuitive FP "junk" propagation. Specifically defined order-of-operations and explicit type coercion addressed the edge cases by forcing FP precision rather than deferring to VBA's evaluation rules.

The numerical precision problem showed up after thousands of entity histories matched, and only happened with an incredibly small input data set during testing. Thankfully, the inconsistencies did **not** occur in Clojure (i.e., moving to Clojure was a "Good Thing"). The cause of the issue was surprising to us, as we verified that the cause existed only in VBA. We were unable to replicate the problem in any programming language at use at CAA for modeling or other data analysis applications, to include R, Python, Scheme, Common Lisp, Julia, Mathematica, and **Mat**rix **Lab**oratory (MATLAB).

3.3 Ordering Issues

Despite the apparent determinism in MARATHON, certain platform-specific implementation details – primarily for associative data structures – presented differences in the output that required explicit controls. The fundamental associative data structures in VBA and Clojure – Dictionary and Persistent HashMap (aka Hash-Array Mapped Trie) – make no guarantees on the order in which key/value pairs are stored. They are "unordered" containers. Any runtime ordering – such as the result of obtaining the keys of the data structure – is likely to be different between data structures.

Differences in runtime ordering affect many situations where concurrency is involved in the face of implicit ordering:

- Order of unit entity updates on the same day.
- Order of demand fill where priority is identical - or demand category / SRC is different.
- Order of deployable units where all other criteria are equal.
- Order of demand activation.

Without an explicit ordering mechanism to control for differences between both data structures, the arbitrary ordering could (and did) cause mismatched simulation history – leading to false positives for differences. For example, "Unit1 may have a recorded deployment textually before Unit2 for effectively arbitrary reasons, despite the deployment entries being consistent between both M3 and M4."

We addressed implicit ordering by adding special ordering constraints to ensure consistency for verification runs. In particular:

- All else being equal, unit indices (integer values) serve as a final sorting criterion for selecting deployable units.

- We pre-processed demand record data to establish a total ordering of the demand, eliminating different fill orderings.

3.4 Model Patches

As expected, M4 bore the brunt of corrective patches (~ 40 corrective commits). However, in some cases, M4 actually demonstrated correct behavior, leading to the discovery of subtle errors in the legacy M3 implementation. During the course of comparative verification, M3 required a few patches to aid in comparisons, as well as correct several previously unknown errors.

Consult APPENDIX D for a manifest of respective M3 and M4 patches.

(THIS PAGE INTENTIONALLY LEFT BLANK)

4 RESULTS

Overall, the MARV study achieved its verification objectives. In particular:

- M3 and M4 model output matched for all capacity analysis cases that we ran for TAA 20-24.
- M3 and M4 model output matched for all requirements analysis cases that we ran for TAA 20-24.
- Tracked down numerous subtle bugs in both versions (M3 – 7 bugs, M4 – 38 bugs) and converged on consistency.
- We added 10 general regression tests to the existing M4 test suite, in addition to hundreds of large-scale regression tests present in TAA data, strengthening current and future confidence in the model implementation.

The methodology limited the verification to comparable cases from M3 out of practicality. While M4 can automatically perform many more verification cases, we lacked the ability to compare against the same number of M3 cases due to the relatively inefficient case generation for M3. In this comparative verification, M3 case generation was the bottleneck.

To our surprise, we found no authoritative guide to verification. Given the variability between different models, and the "custom" nature of the entire simulation domain, the absence of an authoritative, universal guide retrospectively makes sense. Consequently, we drew inspiration from other realms like Software Engineering for the basis of our verification methods.

As an effectively third-class language, unsupported by Microsoft, the default implementation of VBA has systemic technical problems that Microsoft is unlikely to address. These problems include flawed basic numeric routines that defy both reason and the official documentation, and less-fundamental technical issues like a leaking garbage collector. Despite historical claims of portability and purported operations research/systems analysis familiarity, VBA is a less-than-desirable target for simulation development.

In contrast to the limitations of Excel / VBA, the JVM / Clojure / Git combination delivered a sizeable technical advantage for M4 development and testing. Version control, automated regression testing, and the Clojure programming language provided a tight interactive verification feedback-loop. The ability to quickly experiment, test, change, and retest enabled rapid problem discovery and confident, lasting repairs without a loss of version control or the ability to reason about changes to the source code.

When possible, comparative verification is an effective means for building confidence in a large-scale simulation port. To perform comparative verification effectively, the analysis must account for a host of factors:

- Numerical routines must be consistent between platforms.
- Undefined behaviors – like unordered hash-based data structures – must be explicitly controlled or verification metrics insensitive to order should be chosen.

- In order to gain confidence in results of complex deterministic simulations, dynamic verification mechanisms – event logs, interactive simulations, REPLs, visualization, textual post-processing, runtime assertions, and regression are essential.
- A team consisting of geographically separated members required a collaboration mechanism such as the Defense Collaboration Services (DCS) for remote real-time communication and distributed development tooling (e.g., Git).

5 RECOMMENDATIONS

CAA modelers should avoid the singular "it's verified" mentality, and embrace notions of regression testing and continuous integration from Software Engineering. We should continually verify the MARATHON model (i.e., M4 and beyond) via regression and unit tests.

Independent verification offers the possibility for further strengthening confidence in the model implementation. For example, the Naval Postgraduate School data farming cell was a simulation partner in 2014. The technical limitations of M3 precluded fully exploiting their students/faculty/expertise at the time, but that no longer holds true with M4. CAA should participate in external verification of M4 with other DoD agencies and academia.

CAA should exploit the enhanced technical capabilities of M4 in a follow-on effort to strengthen verification results further. Given the more efficient case generation and the ability to distribute simulation runs, generative testing via large-scale design of experiments (e.g., Nearly Orthogonal Latin Hypercube or adaptive experimental design methods) looks like a promising, general-purpose methodology for continual verification efforts. Industry offers another example via chaos – or "fuzz" - testing (à la the Netflix Chaos Monkey testing framework).

Given that M3 and M4 provide comparable output, M4 should be the model of record for all force structure analyses as of July 2017.

(THIS PAGE INTENTIONALLY LEFT BLANK)

APPENDIX A PROJECT CONTRIBUTORS

A-1 PROJECT TEAM

Project Directors:

Rick Hanson, FS

Dr. Jordan Adams, FS

Team Members:

Tom Spoon, FS

Other Contributors:

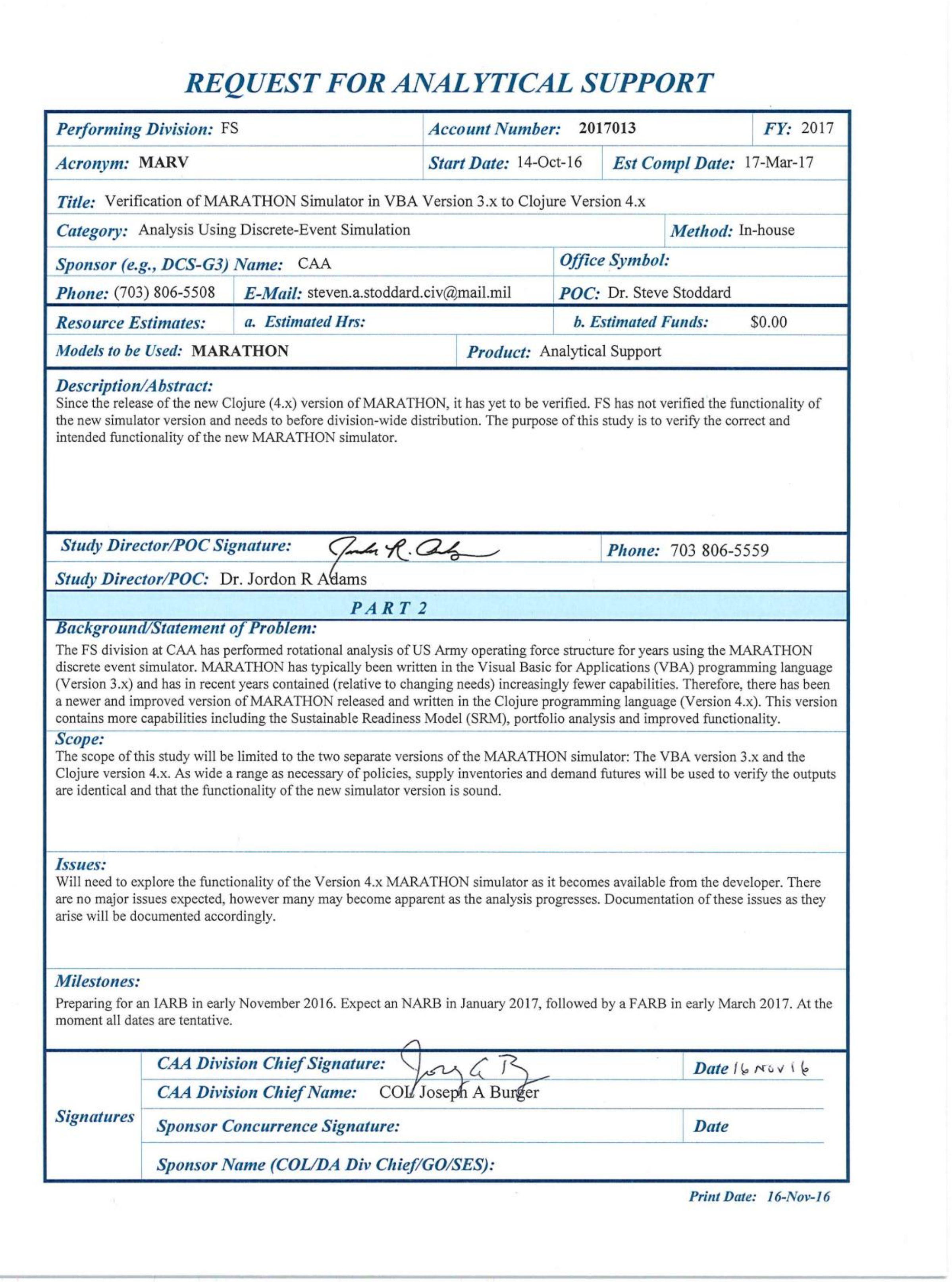
Craig Flewelling, FS

A-2 PRODUCT REVIEWERS

Mr. Russell Pritchard, Quality Assurance

(THIS PAGE INTENTIONALLY LEFT BLANK)

APPENDIX B REQUEST FOR ANALYTICAL SUPPORT



REQUEST FOR ANALYTICAL SUPPORT

Performing Division: FS	*Account Number:* 2017013	*FY:* 2017
Acronym: MARV	*Start Date:* 14-Oct-16	*Est Compl Date:* 17-Mar-17

Title: Verification of MARATHON Simulator in VBA Version 3.x to Clojure Version 4.x

Category: Analysis Using Discrete-Event Simulation — *Method:* In-house

Sponsor (e.g., DCS-G3) Name: CAA		*Office Symbol:*
Phone: (703) 806-5508	*E-Mail:* steven.a.stoddard.civ@mail.mil	*POC:* Dr. Steve Stoddard
Resource Estimates:	*a. Estimated Hrs:*	*b. Estimated Funds:* $0.00
Models to be Used: MARATHON		*Product:* Analytical Support

Description/Abstract:
Since the release of the new Clojure (4.x) version of MARATHON, it has yet to be verified. FS has not verified the functionality of the new simulator version and needs to before division-wide distribution. The purpose of this study is to verify the correct and intended functionality of the new MARATHON simulator.

Study Director/POC Signature: [illegible] — *Phone:* 703 806-5559

Study Director/POC: Dr. Jordon R Adams

PART 2

Background/Statement of Problem:
The FS division at CAA has performed rotational analysis of US Army operating force structure for years using the MARATHON discrete event simulator. MARATHON has typically been written in the Visual Basic for Applications (VBA) programming language (Version 3.x) and has in recent years contained (relative to changing needs) increasingly fewer capabilities. Therefore, there has been a newer and improved version of MARATHON released and written in the Clojure programming language (Version 4.x). This version contains more capabilities including the Sustainable Readiness Model (SRM), portfolio analysis and improved functionality.

Scope:
The scope of this study will be limited to the two separate versions of the MARATHON simulator: The VBA version 3.x and the Clojure version 4.x. As wide a range as necessary of policies, supply inventories and demand futures will be used to verify the outputs are identical and that the functionality of the new simulator version is sound.

Issues:
Will need to explore the functionality of the Version 4.x MARATHON simulator as it becomes available from the developer. There are no major issues expected, however many may become apparent as the analysis progresses. Documentation of these issues as they arise will be documented accordingly.

Milestones:
Preparing for an IARB in early November 2016. Expect an NARB in January 2017, followed by a FARB in early March 2017. At the moment all dates are tentative.

Signatures	*CAA Division Chief Signature:* [illegible]	*Date* 16 Nov 16
	CAA Division Chief Name: COL Joseph A Burger	
	Sponsor Concurrence Signature:	*Date*
	Sponsor Name (COL/DA Div Chief/GO/SES):	

Print Date: 16-Nov-16

(THIS PAGE INTENTIONALLY LEFT BLANK)

APPENDIX C BIBLIOGRAPHY

United States Department of Defense, MIL-STD-3022 w/CHANGE 1, *Department of Defense Standard Practice: Documentation of Verification, Validation, and Accreditation (VV&A) for Models and Simulations*, 5 April 2012.

Defense Modeling & Simulation Coordination Office, *M&S VV&A Recommended Practices Guide*, 31 January 2011.

Defense Modeling & Simulation Coordination Office, RPG Reference Document*, V&V Techniques,* 15 August 2001.

Headquarters, Department of the Army, DA Pam 5-11, *Verification, Validation, and Accreditation of Army Models and Simulations*, 30 September 1999.

Headquarters, Department of the Army, AR 5-11*, Management of Army Modeling and Simulation,* 30 May 2014.

Andreas Tolk, *Engineering Principles of Combat Modeling and Distributed Simulation*, John Wiley & Sons, Inc., 2012.

Don Hodge, *A Practical Introduction to Validation, Verification, and Accreditation*, 84th MORS Symposium, 20 June 2016.

(THIS PAGE INTENTIONALLY LEFT BLANK)

APPENDIX D MODEL PATCHES

D-1 M3 Patches / Errors

(M3-79)

Added unit-index as bottom-line sorting criterion.

(M3-80)

Inconsistent FP results when computing normalized dwell, leading to non-reproducible sorting criteria.

Implemented `floattrunc()` function for cross-platform consistent decimal truncation.

(M3-81)

Discovered and fixed another numerical stability error that was creating sparse off-by-one errors when projecting between policies during policy-change states; used `floattrunc`.

Ensured follow-on units are released in-between fill phases, rather than end-of-day, to allow follow-on-eligible units to re-enter into general availability.

M3 inadvertently re-processed policy changes for some NonBOG units going into re-entry, causing a cycle-time projection that set the unit's cycle time back by exactly 1 day due to truncation.

Fixed M3 error with compo-preference not weighted corrected during sort criteria, leading to incorrect sorting order for units with components outside of the specified component.

(M3-82)

Discovered that VBA's implement of `Fix()` is error-prone for rare corner cases. Rewrote `floattrunc()` to enforce specific order of operations and type coercions that rectify this non-obvious platform bug.

Fixed the problematic NonBOG supply relaxation and an implicit error in assuming zero NonBOG supply without the presence of deployable supply. NonBOG supply no longer dependent on deployable supply.

(M3-83)

Now enforcing initialization erring out if any non-positive values get to demand creation, to prevent demands with zero quantity. Additionally, we have scoping rules in place akin to M4, so that prior to instantiating entities, we actually scope-out invalid records.

D-2 M4 Patches / Errors

Patched lifecycle distribution to be consistent with M3.

Added extra sorting consistency based on unit-index.

Correcting behaviors related to policy / withdraw.

Fixing follow-on and parametric recovery problems.

De-duplicating follow-on releases.

Got recovery, re-entry, and policy change working; working on deferred policy changes.

Fixed policy wait error, max utility working.

Fixed policy change for same positions.

Fixed overenthusiastic overlapping state changes.

Adding in NonBOG infrastructure.

Added FP truncation function.

Corrected post-NonBOG re-entry with deferred policy.

Fixed policy-change induced cycle termination.

Corrected source-first error creating demand entities.

Corrected non-scoping of supply and demand.

Fixed missing follow-on component, scientific numbers.

Added recursive guard for policy changes.

Patched non-bog query for M3-82 consistency.

Corrected minor input validation discrepancy.

APPENDIX E ACRONYMS

AAES	ARFORGEN Availability Exploration Study
ARFORGEN	Army Force Generation
BOG	Boots On Ground
CAA	Center for Army Analysis
DA	Department of the Army
DCS	Defense Collaboration Services
DoD	Department of Defense
FS	Force Strategy (Division of CAA)
G-3/5/7	Office of the Army Deputy Chief of Staff, Operations, Plans, and Training
IDE	Integrated Development Environment
JVM	Java Virtual Machine
M&S	Modeling and Simulation
M3	MARATHON, version 3
M4	MARATHON, version 4
MARATHON	Modeling Army Rotation At Home Or Not
MATLAB	Matrix Laboratory
MARV	MARATHON Verification (this study)
MORS	Military Operations Research Society
NCFA	National Commission on the Future of the Army
REPL	Read-Evaluate-Print Loop
RPG	Recommended Practices Guide
SRC	Standard Requirements Code
SR	Sustainable Readiness
TAA	Total Army Analysis
V&V	Verification and Validation
VBA	Visual Basic for Applications
VV&A	Verification, Validation and Accreditation

(THIS PAGE INTENTIONALLY LEFT BLANK)